**Documentation - TVM MetaWareNN Backend**

**Purpose:**

The purpose of this document is to give an overview of MetaWareNN Backend implementation and to explain workflow from the TVM framework.

**Overview:**

Machine Learning models are being loaded using the ML frameworks. The execution of the models is happening in the host environment (x86 / ARM platform). In order to enable graph execution in other platforms/devices, these ML frameworks added specific implementations to convert ML models to graph Intermediate Representation (IR) and then use the IR to execute the graph in expected devices(DSP, etc).

In TVM, many backends like TensorRT, Vitis AI, Ethos-N, Core ML, etc have unique implementations to run the executable graph. TVM has a backed specific implementation to check for the supported nodes in the graph and generate a TVM Relay compiled file from the ONNX / Caffe / Tensorflow / PyTorch / MXNet framework models. TVM will load the common runtime file into the backend specific implementation and the backend will convert into its own format to run in its device. Similarly, to execute the graph in Synopsys hardwares, we have added MetaWareNN Backend in the TVM framework to generate MetaWareNN Executable Graph.

**TVM - MetaWareNN Repository:**

* Forked [TVM](https://github.com/apache/tvm.git) repository from commit id [f64ddff4977cbb3edc8391a5ee4802e4870846c2](https://github.com/SowmyaDhanapal/tvm/commit/f64ddff4977cbb3edc8391a5ee4802e4870846c2) created a [metawarenn\_dev](https://github.com/SowmyaDhanapal/tvm/tree/metawarenn_dev) branch to incorporate MetaWareNN Backend related code changes.
* Added [MetaWareNN.cmake](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/cmake/modules/contrib/MetaWareNN.cmake) & included MetaWareNN cmake in the [common CMakeLists.txt](https://github.com/SowmyaDhanapal/tvm/blob/79a107050bb19ea9826d921073e5a3838d6e3485/CMakeLists.txt#L397) & [config.cmake](https://github.com/SowmyaDhanapal/tvm/blob/20b9af5d901e32b399aae9d766a7a44117a4210b/cmake/config.cmake#L205) file to compile backend files in TVM
* Added Python & Necessary C files in the TVM Framework to handle the MetaWareNN Backend - [Initial Commit](https://github.com/SowmyaDhanapal/tvm/commit/f235d66ddb9ad6b41a8e47ee4f16ad227e37b20b)
* Added code changes to generate high level MetaWareNN Graph, apply graph transformations(passes) & generate low level MetaWareNN Graph and create InferenceEngine & ExecutionContext using [metawarenn\_lib](https://github.com/SowmyaDhanapal/metawarenn_lib/tree/onnx_conversion)
* **FLAGS to Control Code Flow**
  + By default, *INFERENCE\_ENGINE* is disabled which will create output ONNXProto directly from MWNNGraph and dumps the ONNX model to *op\_onnx\_models*
  + Enable *INFERENCE\_ENGINE* flag to convert MWNNGraph to ExecutableGraph and cache it to EXEC\_DUMPS path and then create Inference Engine & Execution Context and finally dumps the output ONNX model in *op\_onnx\_models*
  + Enable *INVOKE\_NNAC* flag to generate a MetaWareNN Graph proto from the low level MetaWareNN Graph and serialize it to a binary file. EVConvert python module which is integrated in MetaWareNN library will generate a Caffe prototxt & Caffemodel from MetaWareNN binary proto file, and finally evgencnn executable will generate a EV binary using the Caffe files
    - [Note] : INVOKE\_NNAC flag is Outdated and not tested after MWNNGraph update to ONNX format

**Trigger Inference with MetaWareNN Backend:**

* [Register](https://github.com/SowmyaDhanapal/tvm/blob/79a107050bb19ea9826d921073e5a3838d6e3485/src/relay/backend/contrib/metawarenn/codegen.cc#L155) the MetaWareNN backend codegen to the TVM registration API
* Created [partition for MetaWareNN](https://github.com/SowmyaDhanapal/tvm/blob/79a107050bb19ea9826d921073e5a3838d6e3485/metawarenn_inference/mwnn_inference.py#L15) backend for the input model and build & run the same in the [inference script](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/mwnn_inference.py)

**Build subgraph from TVM IR Module and its execution:**

[python/tvm/relay/op/contrib/metawarenn.py](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/python/tvm/relay/op/contrib/metawarenn.py) - Parse the MetaWareNN supported operators from the Json file & registers it for MetaWareNN Backend and Partition & annotate the subgraphs based on the supported operators

[src/relay/backend/contrib/metawarenn/codegen.cc](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/src/relay/backend/contrib/metawarenn/codegen.cc) - The external compiler

/ codegen tool takes a Relay expression/module(TVM) and compiles it into a runtime module(Backend) by serializing the expression to a json representation. No backend specific files are required here as we are handling the same during runtime

[src/runtime/contrib/metawarenn/metawarenn\_json\_runtime.cc](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/src/runtime/contrib/metawarenn/metawarenn_json_runtime.cc) - Build the backend specific MetaWareNN Graph from the JSON Representation. Apply the optimization passes & generate the MetaWareNN Executable network from the low level MWNNGraph. Create Inference Engine and Execution Context using Executable Graph. Fill the backend input buffers from TVM, Execute the network in backend using Execution Context and Fills the TVM output buffers from backend

[mwnn\_inference.py](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/mwnn_inference.py) - Inference script which partitions the graph in the MetaWareNN Backend and triggers the build & run call for the input MobileNetv2 model

[metawarenn\_inference/test\_regression\_onnx.py](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/test_regression_onnx.py) - Python Regression script to run inference for all the ONNX models in [models.txt](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/models.txt) using [onnx\_inference.py](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/onnx_inference.py)

[metawarenn\_inference/test\_regression\_tflite.py](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/test_regression_tflite.py) - Python Regression script to run inference for all the TFLite models in [tflite\_models.txt](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/tflite_models.txt) using [tflite\_inference.py](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/tflite_inference.py)

[metawarenn\_inference/test\_regression\_quantized\_tflite.py](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/test_regression_quantized_tflite.py) - Python Regression script to run inference for all the quantized TFLite models in [tflite\_quantized\_models.txt](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/tflite_quantized_models.txt) using [quantized\_tflite\_inference.py](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/quantized_tflite_inference.py)

**MetaWareNN Library Documentation:**

Refer to [this document](https://github.com/SowmyaDhanapal/metawarenn_lib/blob/metawarenn_dev/docs/Documentation-MetaWareNNLibrary-Structures.docx) for the details on MetaWareNN Library class structures, Optimizers, Serializations & Inference Engine creation for Subgraphs

**Clone the Repository:**

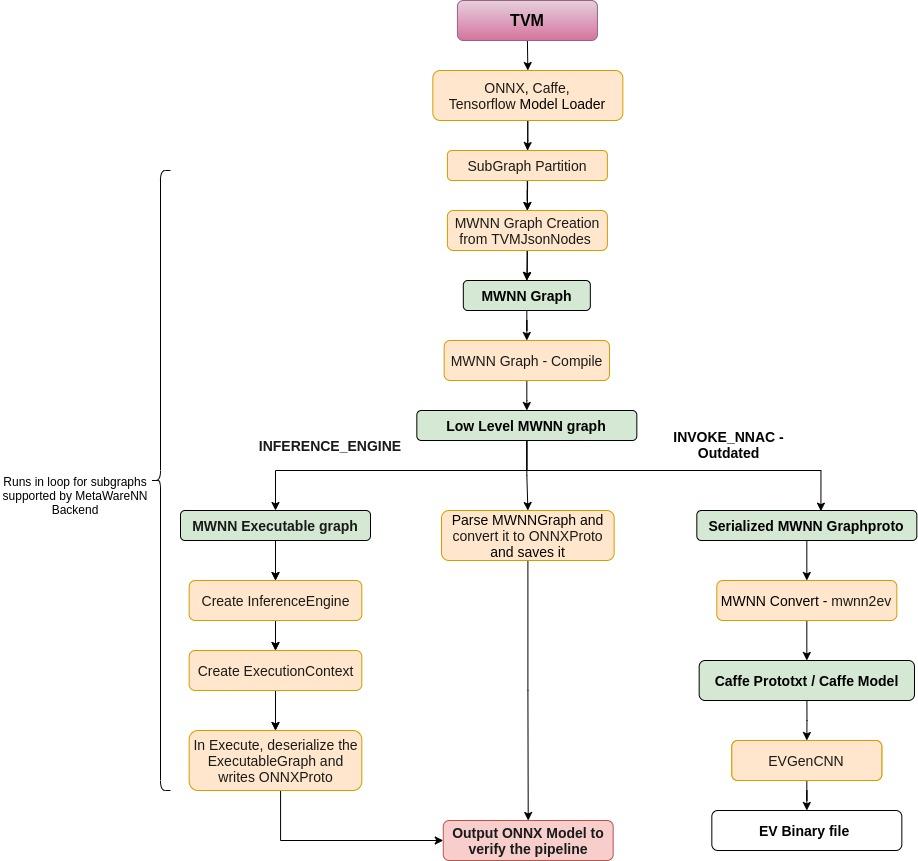
1. *git clone --recursive https://github.com/SowmyaDhanapal/tvm.git tvm*
2. *cd tvm*
3. *git checkout metawarenn\_dev*
4. *git submodule sync*
5. *git submodule update*
6. *sudo apt-get update*
7. *sudo apt-get install -y python3 python3-dev python3-setuptools gcc libtinfo-dev zlib1g-dev build-essential cmake libedit-dev libxml2-dev*

**Modifications to make in the repository:**

*Refer this* [*README.md*](https://github.com/SowmyaDhanapal/tvm/blob/metawarenn_dev/metawarenn_inference/README.md) *file to get the detailed steps to compile and run the inference for sample models*

**Flow chart:**

The workflow of the MetaWareNN Backend in TVM has been depicted in the following diagram,

****